Compact Code Formatting Style

The code formatting style used for Tcl programs as reflected on the man pages and elsewhere is similar to C code formatting. I propose an alternative, more compact style: CCFS. Not because I want it to be *the* standard, but rather to share the idea. Read the Tcl Style Guide as the first source of how to write well behaved programs!

After reading (once again) '1% the code' [L1] and while writing (once again) a medium size Tcl application, I started to think that there are too many braces in the files. Python [L2] does without them, however Python lacks nice syntax at other points, I won't discuss this here though.

CCFS rules:

no lonely braces on a single line, except:

1. when a namespace, or a proc larger then 24 lines (buh) ends, or
2. for delimiting data

- braces in if {expr} only when it is an expression
- braces around arguments only if there are more then one

Some recommendations:

- rather return, continue, break then else
- don't while, foreach, for, if you can avoid it - recurse

Illustrating with an example:

```
proc ccfs param {
    if !$param return
    puts "we are in"

    if [llength $param] {puts "and we've got a list"}

    switch -- [lindex $param] {
        stop {return}
        continue {
            # test the second parameter
            set test [lindex $param 1]}
        default {puts "don't know what todo with: $param}}}
   if {$test eq "stop"} {
       puts "we stopped on the second param"}}
```

Observations:

- If a line is empty, it clearly denotes that something new is going to happen.

- Of course I use a syntax aware editor to get the indenting and the parent matching right
- Conditional are: either flags - $var, results - script, or expressions - {expr}, which is visually conveyed by the CCFS
- Some say (A question of style) that run time is slightly longer; I don't care! (If you need it fast, postprocess the code to insert the braces).

Mhm.. if/then/else is bad, however, sometimes we need it though. If an if/else branch is too long to fit into the same line, you have to decide upon formatting. What about?

```
 Traditional       CCFS 1: tame       CCFS 2: else = } {             CCFS 3: wild
                                      you might spare on more line    i like this
one
                                      if '..else..' is short          it gets the
} out of sight

 if {cond} {       if {cond} {        if {cond} {                     if {cond} {
     ..then..          ..then..           ..then..                        ..then..
} \
 } else {          } else {           } {                             else {
    ..else..           ..else..}          ..else..}
..else..}
 }                 continuation..     continuation..
continuation..
 continuation..
```

Here is some real code, cut&paste from ttp.tcl from TTP. Please don't try to understand the code, just skim over the text to see the syntactic patterns:

```
  CCFS                                             extreme example of standard
syntax
  ..inside a namespace...
    proc tcl {args} {                              proc tcl {args} {
        variable state                                 variable state
        if [llength $args] {                           if {[llength $args]} {
            switch $state {                                switch $state {
                parse {set state tclline}}                     parse {
            catch {eval $args} result                              set state
tclline
            return $result                                     }
        } else {                                           }
            switch $state {                                catch {eval $args}
result
                parse {set state tclstart}                 return $result
                tcl {set state tclend}}}               } else {
        return}                                            switch $state {
                                                               parse {
    # cmd: preprocess lines instead of subst                       set state
tclstart
    proc cmd {args} {                                          }
        variable state                                         tcl {
        variable cmdLine                                           set state
tclend
        switch $state {                                        }
            parse {                                         }
                set cmdLine $args                      }
                set state cmdstart}}}                  return
                                                   }
    namespace export out parse skipline -- literal tcl cmd
 }                                                 # cmd: preprocess lines instead
of subst
                                                   proc cmd {args} {
 namespace import ::ttp::*                             variable state
                                                       variable cmdLine
 proc stamp {} {                                       switch $state {
    set host ""                                            parse {
    if [info exists ::env(HOST)] {set host $::env(HOST)}       set cmdLine $args
    if [info exists ::env(HOSTNAME)] {set host $::env(HOSTNAME)} ..............
    if {$host eq ""} {catch {exec hostname} host}          }
 ...the procedure continues here...                    }
```

Loops

'for' and 'while' use expressions for iteration, however for reasons explained elsewhere you **must** enclose the expression within braces, which is ugly. Use the intrinsic list processing of the Tcl 'proc' instead. The command line parser of TTP is an example for this. Iff:

- the iteration is not very deep
- does not get called all the time

The notational and expressive elegance of recursion is almost every time better then looping (says I, but .. who am I to say this?).

'foreach' is an exception, especially if you iterate over more then one variable.

Ugly example:

```
proc printlist args {
   while {[llength $args]} {
       puts [lindex $args 0]
       set args [lreplace $args 0 0]}}
```

Good looking:

```
proc printlist {item args} {
    puts $item
    if [llength $args] {eval printlist $args}}
```

Of course this is a very constructed example since the following is **the** way to do it:

```
proc printlist args {foreach item $args {puts $item}}
```

However i hope to illustrate the point of using 'proc' and 'args' for list iteration. For counting stuff consider:

```
proc forloop i {
     if $i {puts $i;  forloop [incr i -1]}}
```

Oh.. this counts down and stops with '1'! Ahem, does that really matter? Yes! Then use:

```
proc forloop {i n} {
    if $n {puts $i;  forloop [incr i 1] [incr n -1]}}
```

See Tail call optimization for more on recursion and: program language specialists please jump in.

LEG

RLH That code is hard to read. Much more so than regular Tcl syntax style.

jcw - Check out an indentation syntax for Tcl ...

LEG - in fact i read that before making up this page. an indentation syntax for Tcl however changes the syntactic rules, CCFS does not. I was rather inspired by Lisp than Python. However i would like to see a back-and-forth code reformatter between CCFS and standard Tcl Syntax: would your programm be the right tool to take as a start?